Debugger Menus 31

This chapter describes each of the commands in the source-level **Debugger** menus. It is organized by menu, from left to right along the menu bar. Within each menu, commands are described in the order in which they appear in the menu.

Some of the Debugger commands operate on the selected statement. To select a statement, just click its line in the Source window. If there is no selected statement, the commands operate on the current statement.

Contents

The Apple Menu 677
Shortcuts 677

The File Menu 677
Save All 677
Close 677
Close All 677

The Edit Menu 678
Undo 678
Cut 678
Copy 678
Paste 678
Clear 678
Copy To Data 678

The Debug Menu 679
Go 679
Step 679
Step In 679
Step Out 680
Trace 680
Stop 680
Go Until Here 680
Skip To Here 680
Inspector 681

Monitor 681
ExitToShell 681

The Source Menu 682
Clear Breakpoint
Set Breakpoint 682
Clear All Breakpoints 682
Attach Condition 682
Show Condition 682
Edit ‘filename’ 682

The Data Menu 683
Clear All Expressions 683
Set Context 683
Show Context 683
Signed Decimal
Unsigned Decimal
Hexadecimal
Character
Pointer
Address
C String
Pascal String
Floating Point 684
Locked 685
Context Free 685

The Windows Menu 685
projectname 685
filename.c 685
Data 685

The Apple Menu

Shortcuts

This command displays a series of dialog boxes describing some shortcuts that make working with the debugger faster.

The File Menu

*The **File** menu lets you save your current debugging session.*

Figure 31-1 The File menu

Save All

This command saves your current debugging session, including your breakpoints and the contents of your data windows, so the debugger can restore your session the next time you use the debugger. If the Always save session option is on in the Debugging page of the **THINK Project Manager options** dialog, the debugger also saves your session whenever you quit the debugger.

Close

This command lets you close a single array or struct window. You can do the same thing by clicking the particular window's Close box.

Close All

This command lets you close all the array or struct windows at once. You can achieve the same result if you hold down the Option key as you click the Close box of any struct or array window.

The Edit Menu

*The **Edit** menu lets you use the standard editing commands on expressions in the Data window.*

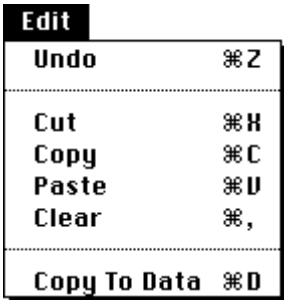

Figure 31-2 The Edit menu

Undo This command undoes any edits you make to an expression in the Data window. Choosing this command is the same as clicking the Deselect button and then selecting the same expression again.

Cut This command deletes the selected text and copies it to the Clipboard. You can't cut text out of the Source window.

Copy This command copies the selected text to the Clipboard.

Paste This command pastes the text in the Clipboard into the current window. You can't paste into the Source window.

Clear This command removes the selected expression from the Data window.

Copy To Data This command is active only when the Source window is the frontmost window. It copies a selected expression from the Source window and pastes it into the Data window, where the expression is compiled. It's up to you to make sure that the text selected in the Source window is a valid expression. **Copy To Data** leaves the expression selected, so you can use some of the formatting commands right away.

The Debug Menu

*Use the **Debug** menu to control the execution of your program. The first six commands in this menu have equivalent buttons in the Source window status panel.*

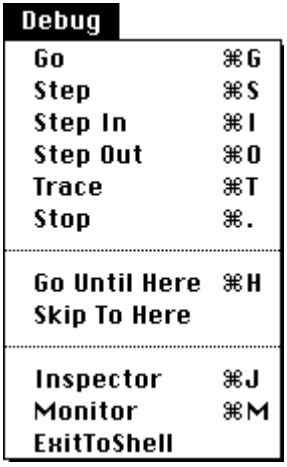

Figure 31-3 The Debug menu

Go

The **Go** command starts your program if it was stopped. Your program runs until you stop it (with the Stop button, for example), until it's about to execute a line with a breakpoint, or until it hits an exception (dividing by zero, for instance). If your application is already running, the **Go** command brings it to the foreground.

Step

The **Step** command goes on to the next statement marker in the current function. If you're at the end of a function, **Step** returns to the calling function. Use **Step** when you want to follow the execution within a function without falling into other functions. (Technically speaking, **Step** skips over JSRs.)

Step In

The **Step In** command executes your program until the current statement arrow falls into a function. **Step In** is useful when you want to skip over a set of assignments or Toolbox traps, in order to fall into the next function call. If **Step In** reaches the last statement of the current function without falling into another function, it stops immediately after the current function returns.

Step Out

The **Step Out** command executes your program until the current statement arrow falls out of the current function. This operation can be slow if there's a lot left to do, but it's a sure way of leaving the current routine. A faster way is to use the **Go Until Here** command or to set a temporary breakpoint at the last diamond in the function.

Trace

The **Trace** command executes the current statement. In most cases, the statement indicator goes on to the next statement marker, even if the next statement marker is in another function. The only time it doesn't is when the program counter steps into some code that the debugger doesn't have the source text for. This usually happens when you step into a trap that's not generated inline. So, for a brief period, the current statement arrow isn't really anywhere in your program, but is somewhere in `MacTraps` instead. Though you can't see the current statement arrow, the current function indicator at the lower left tells you which file it's in.

Stop

The **Stop** command stops execution of your program. It works when any debugger window is active, and the Stop button only works when the Source window is the active window. When you press the Stop button you are usually coming out of your call to `GetNextEvent()` or `WaitNextEvent()`.

If your program is not in its event loop, you might not be able to make the debugger the frontmost application. In this case, Command-Shift-Period is the "panic button." Use Command-Shift-Period to stop execution when one of your application's windows is frontmost or when you think it's stuck in a loop. (Command-Shift-Period doesn't work if you're stuck in an infinite loop in ROM, though.)

Go Until Here

The **Go Until Here** starts execution and stops at the selected line. This command is the same as setting a temporary breakpoint at the selected line. Use this command when you want to move through a block of code quickly.

Skip To Here

The **Skip To Here** command changes the program counter to the selected line without executing any intervening code. Use it when you want to skip over code you know to have bugs but that is not crucial to the rest of the program's operation.

Warning

This command can be dangerous. Make sure that you're not skipping code that depends on the rest of the program. For example, it is a very bad idea to skip over initialization routines.

Inspector

This command is available only if you have Symantec C++. It launches the THINK Inspector, which gives you an easy and graphical way to examine your classes and objects while your program is running. The THINK Inspector is described in the *Visual Architect and THINK Class Library Guide*.

Monitor

The **Monitor** command drops you into a low-level debugger. All your registers (including status registers) and low-memory globals will be correct. The PC (program counter) will be somewhere in the source debugger, not in your program. You can still get the value of your PC.

If you're using TMON, the PC is in TMON's `V` register. If you selected an expression in the Data window, its value is in TMON's `N` register.

If you're using MacsBug, your PC is one long word before the current PC. To look at the instructions in your program, type:

```
DM PC-4
IL @.
```

Note

If you don't have a low-level debugger installed, don't use the **Monitor** command.

ExitToShell

This command aborts the source-level debugger. You should use your application's **Quit** command to quit the debugger. Use **ExitToShell** only if you can't use your application's **Quit** command.

The Source Menu

*The **Source** menu contains commands for working with the Source window.*

Figure 31-4 The Source menu

Clear Breakpoint
Set Breakpoint

This command clears or sets the breakpoint at the selected statement. **Clear Breakpoint** is available when you have already set a breakpoint at the selected statement. **Set Breakpoint** is available when you haven't yet set a breakpoint .

Clear All Breakpoints

This command clears all the breakpoints in the project.

Attach Condition

Use the **Attach Condition** command to attach an expression in the Data window to a breakpoint to create a conditional breakpoint. To set a conditional breakpoint:

1. Set a breakpoint by clicking the statement marker diamond.
2. Click the line to select it.
3. Click an expression in the Data window.
4. Choose **Attach Condition** from the **Source** menu.

Show Condition

Use the **Show Condition** command to display the condition attached to a conditional breakpoint. If the selected statement has a conditional breakpoint (a gray diamond), the attached expression in the Data window is highlighted.

Edit '*filename*'

This command brings THINK Project Manager to the foreground and opens an edit window for the file in the Source window (in this case, the file for the Hello World tutorial program). This is the inverse of the **Debug** command in THINK Project Manager's **Source** menu.

The Data Menu

*The commands in the **Data** menu operate on expressions in the Data window. You can set and show the context of expressions, change their display format, and lock expressions to keep them from being re-evaluated.*

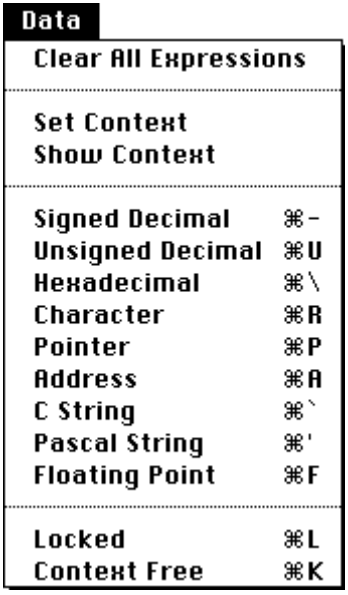

Figure 31-5 The Data menu

Clear All Expressions This command clears the contents of the Data window and closes all your array and struct windows. It's especially useful if you don't want the debugger to restore the contents of your Data window the next time you use the debugger.

Set Context This command makes the selected statement in the Source window the context of the selected expression in the Data window.

Show Context This command highlights the statement that is the context of the expression selected in the Data window.

Signed Decimal
Unsigned Decimal
Hexadecimal
Character
Pointer
Address
C String
Pascal String
Floating Point

These commands control how expressions appear in the Data window. The default format depends on the data type of the expression. The type of the expression also determines what other formats you can use.

The default formats are shown in italics.

Type	Formats Available
integers	*Signed Decimal*, Unsigned Decimal, Hexadecimal, Char
unsigned	*Unsigned Decimal*, Signed Decimal, Hexadecimal, Char
pointers	*Pointer*, Address, Hex, C String, Pascal String
arrays	*Address*, C String, Pascal String
structures	*Address*
unions	*Address*
functions	*Address*
float types	*Floating Point*

This is what the display formats look like:

Format	Example
Signed Decimal	`4523345`, `-1`
Unsigned Decimal	`4523345`, `65535`
Hexadecimal	`0xA09E1487`
Char	`'c'`, `'TEXT'`
C String	`"abcdef\nghi\33"`
Pascal String	`"\pabcdef\nghi\33"`
Pointer	`0x7A7000` or `0x007A7000`
Address	`[] 0x09FE44`, `struct 0x08FC14`
Floating Point	`1961.0102`

The C string and Pascal string formats display nonprinting characters in backslash form. Whenever it can, the debugger uses the built-in escape characters (`\n`, `\r`, `\b`); otherwise it uses `\nnn`, where `nnn` is an octal value.

Of course, you can use type casting to use formats that aren't normally available. For example, if you wanted to see an integer `i` as a C string, you would type this expression: `(char * ) i`.

Locked

The debugger re-evaluates all the expressions in the Data window every time execution stops. To keep an expression from being re-evaluated, select it, then choose **Locked**. When an expression is locked, a small lock icon appears next to it.

To unlock an expression, select the expression, and choose the **Locked** command again.

Context Free

This command makes an expression context free. For example, if you are using one variable name in several routines and you'd like to see the value of the variable when you enter a new routine, instead of seeing a blank space, you can accomplish this by selecting the variable in the Data window, then choosing **Context Free** from the **Data** menu. The debugger marks it with a small arrow, the same arrow that points to the current statement in the Source window. This tells you that the variable's context is the same as that of the current statement.

The Windows Menu

*The commands in the **Windows** menu work on the source Debugger's windows.*

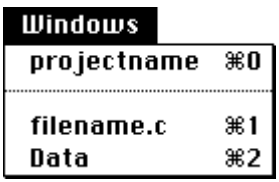

Figure 31-6 The Windows menu

projectname

The real name of this command is the name of your project. Choose this command to bring the THINK Project Manager to the foreground and make the project window the active window.

filename.c

The real name of this command is the name of the file displayed in the Source window. When you choose this command, the Source window becomes the active window.

Data

This command makes the main Data window the active window.

31 Debugger Menus